



THE UNIVERSITY OF  
**WESTERN  
AUSTRALIA**

---

# Shell Arithmetic, Find

---

## Lecture 9

Michael J Wise

# Shell Arithmetic (is not that great)

---

- Shell arithmetic is not very efficient, so if you have any substantial computing to do you can use Awk (coming soon!), or a stand-alone Python, etc program.
- `$ ( ( <expression> ) )` evaluates the expression and returns the result on stdout.

Example:

```
a=1
```

```
a=$(( a + 1 ))
```

```
echo $a
```

2

- Often seen in `while` loops

# Only integer arithmetic

---

|    |                        |
|----|------------------------|
| +  | plus                   |
| -  | minus                  |
| *  | multiplication         |
| /  | integer division       |
| %  | remainder              |
| >> | right shift ( $/2^N$ ) |
| << | left shift ( $*2^N$ )  |
| &  | bitwise AND            |
|    | bitwise OR             |
| ~  | bitwise NOT            |
| ^  | bitwise exclusive OR   |
| && | logical AND            |
|    | logical OR             |
| !  | logical NOT            |

# Demo

---

There was time when programmer productivity was measured in lines of code written.

We'll create a Shell program, `countlines`, to count and report the number of files and the total number of lines across Python files in the current directory

- *have a look at the command: `wc`*

# find

---



- Find is an incredibly useful command
- The task of recursing through directory structures searching for files with specific properties is VERY common

# Find

---

`find [<options>] <path> [<expression>]`

- `find` is given:
  - *one or more directories to search*
  - *an expression specifying the properties of the sought files and what actions to perform when the sought files are found (default `-print` assumed).*
  - *options, which can be used to modify the search, e.g. to limit the depth of the search.*

# Find tests

---

- Some of the more important tests are:

`-name <file pattern>`

The test file matches <file pattern>.

`-type <c>`

The type of the file is as specified by <c>, e.g. `d`

for a directory, `f` for a regular file, `l` for a symbolic link.

`-newer <file>`

*The test file has been accessed more recently than <file> was modified.*

# Find actions

---

- Some of the more important actions are:

`-print`

Prints the full path-name of the file.

`-exec <command>`

Execute `<command>` on each file that survives previous tests. All command-line arguments to find after this are assumed to pertain to the `<command>`, up to a `\;`

- `{ }` refers to the file to which the command is being applied.

`-ok <command>`

- Same as `-exec <command>`, except that user is prompted before `<command>` is executed.
-



# Find actions

---

# list all the files in . And subdirectories

```
find . -print
```

# find every Makefile and call make

```
find . -name "[Mm]akefile" -exec make \; -print
```

# find every file (not dir) and list it

```
find . -type f -exec ls -l '{}' \;
```

# find every rw----- file and make it rw-r--r--

```
find . -perm 600 -exec chmod 644 '{}' \;
```

# Demo

- We'll write `countlinesR`, generalising the `countlines` program from the earlier demo, so that it counts lines from all sub-directories of given directory



Successfully found